

APPENDIX 2

```
/**
 * IPS multithreaded error (rule) processing engine
 * @file error_processor.h
 * @author jmccaskey
 */

#ifndef ERROR_PROCESSOR__H
#define ERROR_PROCESSOR__H

/**
 * Function to process an error node as generated by the rule code.
 * This function should be called as a thread. It is responsible for
 * deciding whether or not an error log entry should be created,
 * creating it if so, and updating the escalation level for rule_monitor table
 * entries.
 */
void error_processor();

#include "error_processor.c"

#endif
```

```

/**
 * IPS multithreaded event processing engine
 * Contains functions responsible for error_log inserts
 * and rule_monitor table updates.
 * @file error_processor.c
 * @author jmccaskey
 */

/**
 * Function to process an error node as generated by the rule code.
 * This function should be called as a thread. It is responsible for
 * deciding whether or not an error log entry should be created,
 * creating it if so, and updating the escalation level for rule_monitor table
 * entries.
 */
void error_processor() {
    MYSQL mysql_connection;
    MYSQL_RES *result;
    MYSQL_ROW row;
    char *sql_query;
    int n;
    int send_notification = 1;

    mysql_thread_init();
    //try the mysql connection
    mysql_init(&mysql_connection);
    if(!mysql_real_connect(&mysql_connection, db_host, db_user, db_pass, db_db, 0, NULL, 0)) {
        flockfile(stderr);
        fprintf(stderr, "%s: Failed to connect to database: Error: %s\n", timestamp,
mysql_error(&mysql_connection));
        funlockfile(stderr);
        pthread_exit(NULL);
    }

    //try to select the database
    if(mysql_select_db(&mysql_connection, db_db)!=0) {
        flockfile(stderr);
        fprintf(stderr, "%s: Failed to select database: Error: %s\n", timestamp,
mysql_error(&mysql_connection));
        funlockfile(stderr);
        pthread_exit(NULL);
    }

    //infinite loop to be exited within when polling is done and nothing is left in the queue
    error_node *errnode;
    while(1) {
        //default back to not sending a notification each loop
        send_notification = 1;

        pthread_mutex_lock(&error_work_queue.mutex);
        while(error_work_queue.c_queue.head==NULL) {
            pthread_mutex_lock(&polling_done_mutex);
            if(polling_done==1) {
                pthread_mutex_unlock(&polling_done_mutex);
                pthread_mutex_unlock(&error_work_queue.mutex);
                //we are finished with everything in the queue and nothing more is

```

coming... time to die

```
        mysql_close(&mysql_connection);
        mysql_thread_end();
        pthread_exit(NULL);
    } else {
        pthread_mutex_unlock(&polling_done_mutex);
        //more errors may be coming... wait for them...
        pthread_cond_wait(&error_work_queue.cond,
&error_work_queue.mutex); //now unlocked... required after we pass
    }

    errnode = (error_node *) queue_get(&error_work_queue.c_queue);
    pthread_mutex_unlock(&error_work_queue.mutex);

#ifdef DEBUG
    flockfile(stdout);
    fprintf(stdout, "%s\n", errnode->message);
    funlockfile(stdout);
#endif

    assert(sql_query = malloc(800));
    n=sprintf(sql_query, 800, "SELECT current_escalation FROM rule_monitor WHERE rule_id =
%d "
        "AND rule_server_id = %d AND monitor_id = %d AND monitor_server_id = %d",
        errnode->rule_id, errnode->rule_server_id, errnode->monitor_id, errnode-
>monitor_server_id);

    //execute query for devices
    if(mysql_real_query(&mysql_connection, sql_query, n)!=0) {
        flockfile(stderr);
        fprintf(stderr, "%s: Failed while attempting to select old escalation level: Error: %s\n",
timestamp, mysql_error(&mysql_connection));
        funlockfile(stderr);
        free(sql_query);
    } else {
        free(sql_query);
    }

    //store results from last query into result
    result=mysql_store_result(&mysql_connection);

    int escalation = 0;
    row=mysql_fetch_row(result);
    if(row==NULL) {
        flockfile(stderr);
        fprintf(stderr, "%s: Couldn't fetch old escalation, assuming it was zero for this
single rule...\n", timestamp);
        funlockfile(stderr);
        mysql_free_result(result);
    } else {
        escalation = atoi(row[0]);
    }
    //now we have the row...
    if(errnode->failed == 0) {
        if(escalation > 0) {
            assert(sql_query = malloc(800));
```

```

        n=snprintf(sql_query, 800, "UPDATE rule_monitor SET
current_escalation = 0 WHERE rule_id = %d "
                                "AND rule_server_id = %d AND monitor_id = %d AND
monitor_server_id = %d",
                                errnode->rule_id, errnode->rule_server_id, errnode->
>monitor_id, errnode->monitor_server_id);
        if(mysql_real_query(&mysql_connection, sql_query, n)!=0) {
            flockfile(stderr);
            fprintf(stderr, "%s: Failed while attempting to reset escalation: Error: %s\n",
timestamp, mysql_error(&mysql_connection));
            funlockfile(stderr);
        }
        free(sql_query);
    } else {
        //this is an ok, and the rule was already ok last period, so don't do
        send_notification = 0;
    }
} else if(errnode->failed == 1) {
    assert(sql_query = malloc(800));
    n=snprintf(sql_query, 800, "UPDATE rule_monitor SET current_escalation = %d
WHERE rule_id = %d "
                                "AND rule_server_id = %d AND monitor_id = %d AND monitor_server_id =
%d", escalation+5,
                                errnode->rule_id, errnode->rule_server_id, errnode->
>monitor_id, errnode->monitor_server_id);
#ifdef DEBUG
        flockfile(stdout);
        fprintf(stdout, "%s\n", sql_query);
        funlockfile(stdout);
#endif
        escalation += 5;
        if(mysql_real_query(&mysql_connection, sql_query, n)!=0) {
            flockfile(stderr);
            fprintf(stderr, "%s: Failed while attempting to increase escalation: Error: %s\n",
timestamp, mysql_error(&mysql_connection));
            funlockfile(stderr);
        }
        free(sql_query);
    }
    if(send_notification == 1) {
        char verdict[6];
        if(errnode->failed == 1)
            strcpy(verdict, "ERROR");
        else
            strcpy(verdict, "OK");

        assert(sql_query = malloc(1000));
        n=snprintf(sql_query, 1000, "INSERT INTO error_log "
                                "(error_log_server_id, rule_id, rule_server_id, monitor_id,
monitor_server_id, "
                                "msg, timestamp, verdict, escalation, notified) "
                                "VALUES (%d, %d, %d, %d, %d, '%s', '%s', '%s', %d, 0)",
                                server_id, errnode->rule_id, errnode->rule_server_id, errnode->
>monitor_id, errnode->monitor_server_id,
                                errnode->message, timestamp, verdict, escalation);

```

```

#ifdef DEBUG
        flockfile(stdout);
        fprintf(stdout, "%s\n", sql_query);
        funlockfile(stdout);
#endif
        if(mysql_real_query(&mysql_connection, sql_query, n)!=0) {
            flockfile(stderr);
            fprintf(stderr, "%s: Failed while attempting to insert into error_log: Error: %s\n",
timestamp, mysql_error(&mysql_connection));
            funlockfile(stderr);
        }
        free(sql_query);
    }
    mysql_free_result(result);
    free(errnode);
}
}

```